Open Research Online

The Open University's repository of research publications
and other research outputs

On the Automated Management of Security Incidents in Smart Space

Journal Item

How to cite:

Alrimawi, Faeq; Pasquale, Liliana and Nuseibeh, Bashar (2019). On the Automated Management of Security Incidents in Smart Space. IEEE Access, 7 pp. 111513–111527.

For guidance on citations see FAQs.

© 2019 IEEE

https://creativecommons.org/licenses/by-nc-nd/4.0/

Version: Version of Record

Link(s) to article on publisher's website:
http://dx.doi.org/doi:10.1109/ACCESS.2019.2934221

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data policy on reuse of materials please consult the policies page.

oro.open.ac.uk

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 23, 2019, accepted August 5, 2019, date of publication August 9, 2019, date of current version August 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2934221

On the Automated Management of Security Incidents in Smart Spaces

FAEQ ALRIMAWI[1], LILIANA PASQUALE[2], AND BASHAR NUSEIBEH[1,3]
[1]Lero—Irish Software Research Centre, University of Limerick, Limerick V94 NYD3, Ireland
[2]Lero, Computer Science Department, University College Dublin, Dublin D04 V1W8, Ireland
[3]School of Computing and Communications, The Open University, Milton Keynes MK7 6AA, U.K.

Corresponding author: Faeq Alrimawi (faeq.alrimawi@lero.ie)

This work was supported in part by the ERC Advanced Grant under Grant 291652 (ASAP), and in part by the Science Foundation Ireland under Grant 13/RC/2094 and Grant 15/SIRG/3501.

ABSTRACT The proliferation of smart spaces, such as smart buildings, is increasing opportunities for offenders to exploit the interplay between cyber and physical components, in order to trigger security incidents. Organizations are obliged to report security incidents to comply with recent data protection regulations. Organizations can also use incident reports to improve security of the smart spaces where they operate. Incident reporting is often documented in structured natural language. However, reports often do not capture relevant information about cyber and physical vulnerabilities present in a smart space that are exploited during an incident. Moreover, sharing information about security incidents can be difficult, or even impossible, since a report may contain sensitive information about an organization. In previous work, we provided a meta-model to represent security incidents in smart spaces. We also developed an automated approach to share incident knowledge across different organizations. In this paper we focus on incident reporting. We provide a *System Editor* to represent smart buildings where incidents can occur. Our editor allows us to represent cyber and physical components within a smart building and their interplay. We also propose an *Incident Editor* to represent the activities of an incident, including —for each activity— the target and the resources exploited, the location where the activity occurred, and the activity initiator. Building on our previous work, incidents represented using our editor can be shared across various organizations, and instantiated in different smart spaces to assess how they can re-occur. We also propose an *Incident Filter* component that allows viewing and prioritizing the most relevant incident instantiations, for example, involving a minimum number of activities. We assess the feasibility of our approach in assisting incident reporting using an example of a security incident that occurred in a research center.

INDEX TERMS Security incidents, smart spaces, incident reporting, incident prioritization, smart buildings.

I. INTRODUCTION

A smart space is a physical or digital environment where humans and technology-enabled systems interact in a connected and intelligent ecosystem. Examples of smart spaces include smart buildings [1]. In a smart building, cyber or physical components can interact with one another. For example, a sensor can communicate a room's temperature to a digital control process that decides whether to turn on the HVAC (Heating, Ventilation, and Air Conditioning) unit to cool/heat the room, or to turn the HVAC off — if it was on.

However, the interplay between cyber and physical components is increasing opportunities for offenders to trigger security incidents and cause harm. For example, in 2014 [2] offenders were able to access the plant network of a German still mill and infect various physical devices connected to the network. This caused significant harm, because the furnace was prevented from shutting down and the alarm system was disabled. Similarly, in 2015 [3], offenders accessed the power grid network of a Ukrainian power grid to infect various devices, such as a serial-to-Ethernet converter and power breakers, with malware. This caused a power outage that affected more than 200,000 customers.

To comply with recent data protection regulations, such as the GDPR, organizations are obliged to report security incidents. Incident reporting [4] is often carried out by writing a document in natural language [5], according to a custom template (e.g., [6]). However, this can be tedious, because

The associate editor coordinating the review of this article and approving it for publication was Mahmoud Barhamgi.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

it requires a manual description of the incident activities, including the vulnerabilities present in the smart space that were exploited during the incident. Incident reporting can introduce errors and omit activities and/or interactions that were relevant to the incident. Although approaches for modeling and visualizing cyber incidents have been proposed, such as attack graphs [7], they only focus on cyber incidents and cannot represent security incidents in a smart space.

Organizations can use incident reports to improve the security posture of the smart spaces where they operate. However, sharing information about security incidents can be difficult, or even impossible, since a report may contain sensitive information. Thus, although reporting of security vulnerabilities is quite common (e.g., as in the Common Vulnerabilities & Exposures database [8]), security incident reports are either kept internally within organizations, are shared partially, or shared only within certain communities, such as the industrial control systems community RISI (Repository of Industrial Security Incidents) [9].

In previous work [10], we provided a meta-model to represent security incidents in smart spaces. We also developed an automated approach to share incident knowledge across different organizations. In this paper, we focus on incident reporting. We provide a graphical *System Editor* to represent smart buildings where incidents can occur. Our editor provides a graphical interface to design the layout of a smart building, drag and drop pre-defined cyber and physical components, and represent their interactions. We also propose an *Incident Editor* that provides a graphical interface to represent the activities of an incident, including —for each activity— the target and the resources exploited, the location where an activity can occur, and the activity initiator. Target, resource, location and activity initiator can be imported directly from a model of a system created using our *System Editor*, or can be created from scratch. The *Incident Editor* also allows validating an incident model to identify issues and possible problems.

Building on our previous work, incidents represented using our *Incident Editor* can be shared and instantiated in different smart spaces, to assess how can they re-occur. However, because incidents can re-occur in many different ways, it would be cumbersome to filter the most relevant incident instantiations among all the possible ones. To address this problem, we propose an *Incident Filter* component that allows viewing and prioritizing the most relevant incident instantiations, for example, by focusing on instantiations involving a minimum number of activities. We demonstrate the feasibility of our approach in assisting incident reporting and prioritization using an example of a security incident that occurred in an actual research center to which we had access. We model the research center rooms, components, and behavior using the *System Editor*. Then, we model the incident activities using the *Incident Editor*. Finally, we use the *Incident Filter* to prioritize incident instantiations. We measure the overhead (time) required for modeling, and discuss the advantages and limitations of our approach.

The paper provides the following contributions. a) We develop an approach to support reporting of incidents in smart spaces. b) Building on our previous work [10], we support sharing and visualization of incident instantiations in different smart buildings. c) Finally, we provide filters to prioritize incidents depending on their number of actions or the components of the smart space that they involve.

The rest of the paper is structured as follows. In Section II, we discuss related work. In Section III, we describe a motivating example to illustrate the current issues with reporting a security incident in smart spaces. In Section IV, we present the architecture of our approach. In Section V, we describe how the various components of our proposed architecture can be used during incident reporting. Then, in Section VI, we illustrate the implications of this work on our previously developed techniques for sharing and assessing security incident knowledge. We demonstrate our approach using a case study in Section VII. Finally, we conclude and discuss future work in Section VIII.

II. RELATED WORK

Incident reports describe the activities performed by offenders. They are usually expressed in natural language and can follow a custom structure. For example, the reports of the German steel mill [2] and the Ukrainian power grid incidents [3] are expressed in natural language and describe various aspects of the incident, such as activities and targeted components. Because security incidents reports are usually written manually, they may be tedious to create and can have errors and omissions. Although data mining techniques (e.g., text mining) have been applied over reports to extract incident information (e.g., actors involved) [11], they require the report to follow a precise structure. Sharing security incident information across organizations can also be difficult, because different organizations can use a different format to represent incidents.

Various approaches have been proposed to represent security incidents. For example, attack graphs [12] represent visually the actions of a cybersecurity incident. However, they do not allow representing other information that can be relevant to the incident, such as the assets targeted and the resources adopted by offenders. STIX (Structured Threat Information Expression) [13] was proposed as a language for representing Cyber Threat Intelligence (CTI). It provides various entities and relationships to represent CTI such as *Threat Actor* entity, which represents an offender, *Malware*, and *uses* relationship, which represents the use of a one entity by an actor. STIX uses TAXII (Trusted Automated Exchange of Intelligence Information) [14] to share incident information. While STIX allows the representation and sharing of CTI between organizations, it does not provide a way to reason automatically about how an incident can re-occur in a different system. Moreover, both attack graphs and STIX do not allow representing incidents in smart spaces, which can involve cyber and physical components and their interaction.

Reports are shared via common repositories, which can be private, i.e. accessed by certain communities, or public. Repository of Industrial Security Incidents (RISI) [9] is a private resource that provides reports about incidents that occurred in industrial control systems. The Common Attack Pattern Enumeration and Classification (CAPEC) [15] catalog is a public repository that provides description of various attack patterns against "cyber-enabled systems". Both RISI and CAPEC provide information expressed in natural language about incidents. Therefore, automation of such resources for analysis (e.g., to assess whether an incident can happen in a specific system) is limited.

Sharing information about an incident mainly revolves around sharing partial information, such as vulnerabilities, via a common repository. For example, the Common Vulnerabilities & Exposures (CVE) [8] resource is a public dictionary of known cybersecurity vulnerabilities in software and devices, which were exploited during a security breach. Based on CVE, the National Vulnerability Database (NVD) [16] attaches various metrics to cybersecurity vulnerabilities such as severity, impact on environment, and interactions required from users. Both CVE and NVD can be analyzed automatically to detect potential incidents in a system. Potteiger et al. [17] proposed an approach for threat modeling and risk assessment that uses the scoring system provided by NVD to quantify vulnerabilities in attack graphs. However, existing approaches only allow sharing information about vulnerabilities, neglecting the activities performed during a security incident.

III. MOTIVATING EXAMPLE

In this Section we use a motivating example to illustrate the issues facing the reporting and sharing of security incidents in smart spaces.

Our example revolves around the ACME company, which operates across three different smart buildings: a *Research Center*, a *Warehouse*, and a *Manufacturing Plant*. This is depicted in Fig. 1. The plan of the 2nd floor of the *Research Center* consists of a *Server Room*, a *Control Room*, and a *Toilet*. The *Server Room* contains a *Fire Alarm*, an air conditioning unit (*HVAC*), and some *Server*s, while the *Control Room* contains a *Workstation*. The whole building is equipped with *Smart Light*s. The *HVAC*, the *Fire Alarm* and the *Smart Light*s communicate with the *Workstation* through the *Installation Bus* network, which adopts the KNX protocol [18].

A security incident occurred in the 2nd floor of the *Research Center*. An offender entered the *Toilet*, and then connected to the *Smart Light* (*SL1*) using a *Laptop*. After that, s/he obtained access to the *Installation Bus* and was able to collect data transmitted over the network (e.g., data exchanged between the *Workstation* and the *HVAC*). This was possible because messages exchanged through the installation bus are not encrypted [19]. The offender then sent a targeted *Malware* (e.g., exploiting the vulnerabilities present in Trane HVACs [20]) to disable the *HVAC*, subsequently causing the *Server*s to heat up. The incident actions are listed at the bottom of Fig. 1.

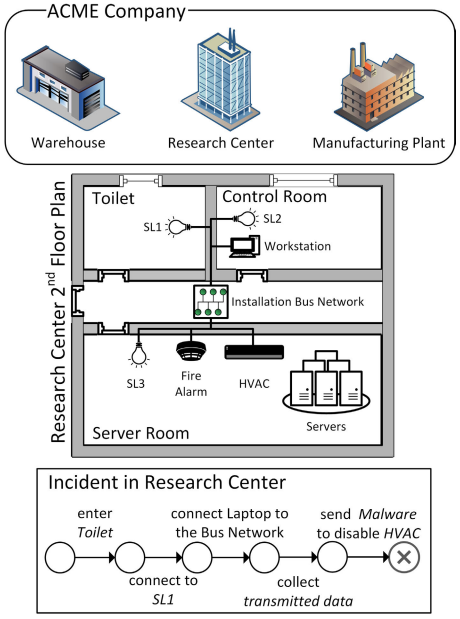

FIGURE 1. The ACME company motivating example.

Upon the discovery of this incident, it was necessary to collect more information, in order to assess how the incident occurred and write an incident report. An incident report should describe the activities performed by an offender, such as *"enter Toilet"* or *"connect Laptop to the Bus Network"*. For each activity it is necessary to specify the location where an activity is performed (*Toilet*), the target components that were exploited (e.g., smart light *SL1*), the resources that where used (e.g., *Laptop*), and who performed the activity (e.g., *Visitor*). Moreover, for each activity it is important to specify *containment* and *connectivity* relationships in the smart space that allowed it to be performed. For example, an offender was able to access *SL1* because it is *contained* in the *Toilet*, which in turn is connected to the *Hallway*. Also, the offender was able to connect his/her *Laptop* to the *Bus Network* by replacing *SL1*, which was initially connected to the network.

However, the reporting process that each organization follows is not standardized [21]. Incident reports may not provide a structure to represent the vulnerabilities exploited by an offender. These vulnerabilities can be brought by cyber and physical components and their interactions in a smart space. Thus, security administrators may overlook, in their report, some characteristics of the smart space that allowed the incident activities to be performed (e.g., connectivity of *SL1* and the *HVAC* to the *Bus Network*).

Security incident reports can be *shared* within the same or a different organization. Before a report is shared, it is necessary to ensure that it does not contain any sensitive information (e.g., network structure) about the system in which the incident occurred [21]. In our example, the report describing

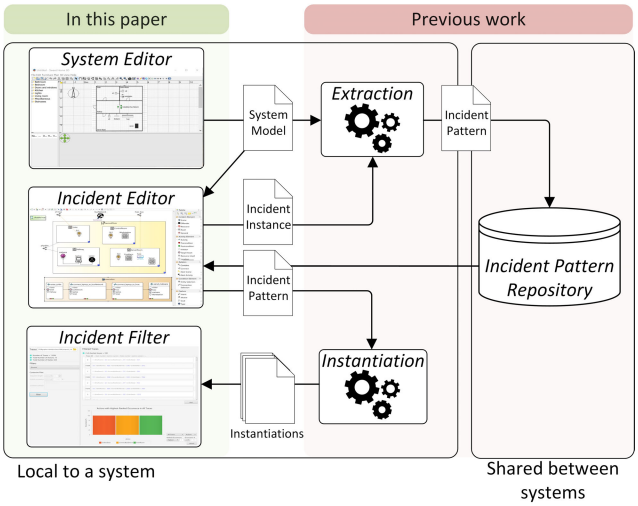

FIGURE 2. The architecture of our approach for incident reporting.

the security incident at the research center can be shared within the same organization to assess how the incident can re-occur in another smart building, such as the manufacturing plant. To achieve this aim, it is necessary to assess how each activity can happen, in a similar way, at the manufacturing plant, depending on its cyber and physical components and their connectivity and containment relationships. However, this is arduous since it requires a security administrator to analyze the incident report manually, and then speculate on all possible ways in which the incident activities can be performed. Additionally, there can be many possible ways in which an incident can occur (instantiations) in a new system, which cannot be captured manually. Also, security administrators may focus on some incident instantiations that are less relevant or likely, overlooking some of the most important ones.

Thus, it is necessary to provide an approach to assist security administrators in reporting security incidents in a more automated and structured form. Incident reports should also allow representing cyber and physical components of a smart space that allowed incident activities to be performed. Finally, when an incident report is shared, it is necessary to support security administrators in visualizing and prioritizing the incident instantiations that are more likely and/or relevant.

IV. ARCHITECTURE FOR INCIDENT REPORTING

We developed an approach to automate reporting of incidents in smart spaces, particularly smart buildings. As shown in Fig. 2, we developed a *System Editor* and an *Incident Editor* to support incident reporting. Building on our previous work [10], we can share incident reports across different organizations and instantiate them onto different smart buildings. We also provide an *Incident Filter* to support visualization and prioritization of incident instantiations in different smart buildings.

- The *System Editor* provides a GUI to represent smart buildings and their dynamic behaviour. As shown in Fig. 3, the *System Editor* includes a *Component Editor*, an *Action Editor*, and a *System Model Builder*. The *Component Editor* extends a graphical designer for smart buildings that we proposed in previous work [22]. This graphical designer builds on SweetHome3D,[1] an open source software tool to draw house plans. SweetHome3D allows creating the physical layout of a building by dragging and dropping pre-defined elements representing, for example, rooms, doors, furniture and so on. Our graphical designer allows representing digital devices (HVACs, computers) and network connectivity within buildings created using SweetHome3D. It also allows one to import the physical layout of a building directly from models compliant with the BIM (Building Information Model) standard [23]. The *Component Editor*, in turn, extends our graphical designer by adding more pre-defined components, such as smart lights and access points, that can be included in the building layout. The *Action Editor* is a graphical designer that allows a security administrator to represent the dynamic behaviour of a smart building. The *Action Editor* allows creating and adding actions that can be performed in a smart building, such as "enter room" or "connect to network". The *System Model Builder* converts the representation of the smart building and its dynamic behavior to a format compliant with the meta-model that we proposed in previous work [10] to represent smart buildings.

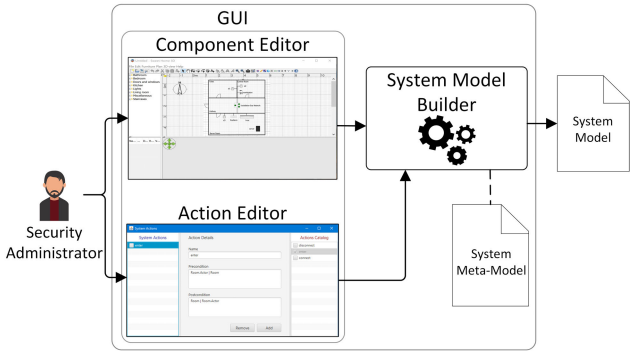

FIGURE 3. *System editor* architecture.

- The *Incident Editor* provides a GUI to model and view incidents in smart buildings. To represent an incident it is necessary to import a model of the smart building where the incident occurred. The *Incident Model Builder* converts the graphical representation of the incident to a format that is compliant with a meta-model, which we proposed in previous work [10] to represent and share incident information.

After a representation of an incident instance is created using the *Incident Editor*, we reuse two automated techniques for sharing incident information: incident extraction and instantiation [10] (see Fig. 2). Incident

[1] http://www.sweethome3d.com/

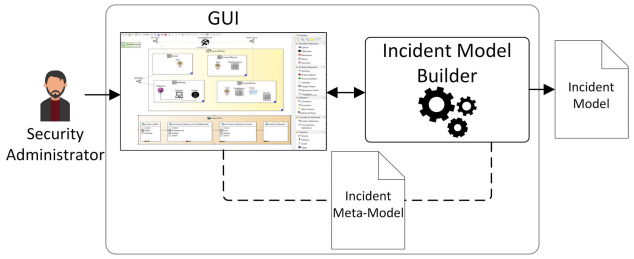

FIGURE 4. ***Incident editor*** **architecture.**

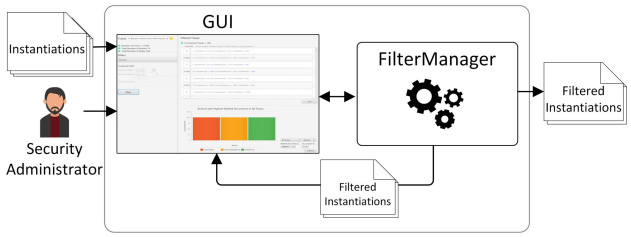

FIGURE 5. ***Incident filter*** **architecture.**

extraction creates a more abstract representation of an incident instance, called *incident pattern*. This does not contain sensitive information which otherwise would be shared. In an *incident pattern*, entities within the smart building referred by the incident activities are only represented using their type (e.g., *SmartLight*), without representing the specific instance name (e.g., *SL1*). Incident activities are also merged following a pre-defined set of activity patterns. Created patterns are stored in an *Incident Pattern Repository*. When a new incident pattern is shared, organizations can access the repository to assess whether and how that pattern can materialize in the smart buildings under their control. Our incident instantiation technique generates all potential instantiations of an incident pattern in a specific smart building.

- The *Incident Filter* provides a GUI to visualize and filter a given set of potential incident instantiations. Filtering prioritizes the most critical incident instantiations depending on different criteria, such as instantiations that have the shortest number of activities, that include specific types of activities, or have activities that target a given asset. As shown in Fig. 5, the *Incident Filter* takes as input all the incident instances generated during the instantiation activity, and allows a security administrator to select a filtering criteria.

  The *FilterManager* component applies the filtering criteria selected, and returns the most relevant incident instantiations, which can be visualized and/or stored — if necessary.

V. TOOL FOR AUTOMATED INCIDENT REPORTING

In this Section we describe in more detail the functionalities implemented by the *System Editor*, the *Incident Editor* and the *Incident Filter* to support automated incident reporting.

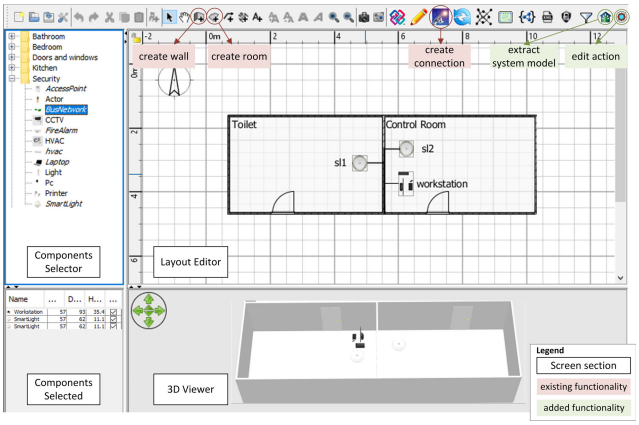

FIGURE 6. ***System editor*** **main screen.**

A. SYSTEM EDITOR

Fig. 6 shows the main panels and functionalities of the *System Editor*. The *Layout Editor* contains the plan of the smart building to be edited. It is possible to create rooms and walls by selecting the corresponding functionalities in the top tab. The *Components Selector* allows selecting the components (furniture, doors, digital devices) that can be contained in each room and can be dragged to the *Layout Editor*. The *3D Viewer* shows a 3D view of the smart building created in the *Layout Editor*. It is also possible to add network connectivity between devices in the building by selecting functionality *create connection* in the top tab. Moreover, function *edit action* allows adding and editing actions representing the dynamic behavior of the smart building. Function *extract system model* is used to generate a system model out of the rooms and components added to a layout. We added the *extract system model* and *edit action* functionalities to the original SmartHome3D tool.

A model of the smart building can be created by performing the following operations.

1) *Create Layout*. The layout of a floor can be created by adding rooms and walls to the *Layout Editor*. For example, the floor layout of the research center considered in our motivating example is shown in Fig. 7.
2) *Add Components*. After creating the rooms, it is possible to drag components (e.g., actor, HVAC) from the *Components Selector* and then drop them into the targeted room. Adding a component to a room establishes a *containment* relation between the component and the room; i.e. the room *contains* the component. In Fig. 7 it is possible to see the components that are contained inside each room of the research center. Network connections between components can also be established by selecting two components to be connected and then clicking on the *create connection* functionality.
3) *Add/Create Actions*. Using functionality *edit action* it is possible to create and/or import a pre-defined set of actions, in order to represent the dynamic

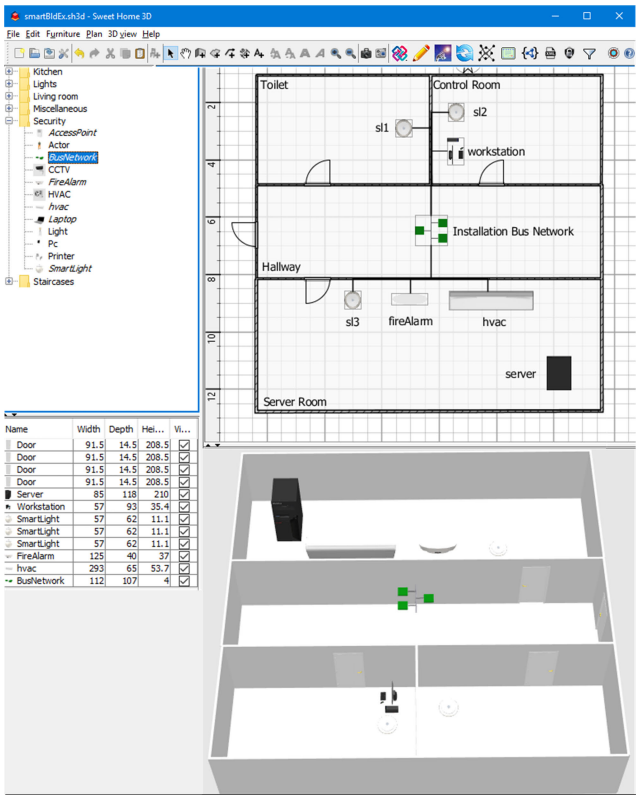

FIGURE 7. Layout of the research center that is modeled using the *system editor*.

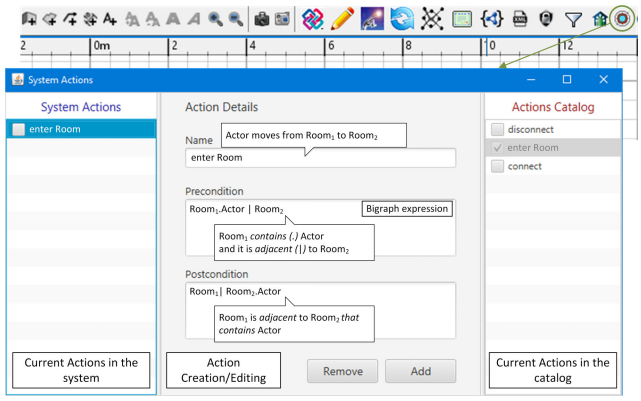

FIGURE 8. The *action editor* screen.

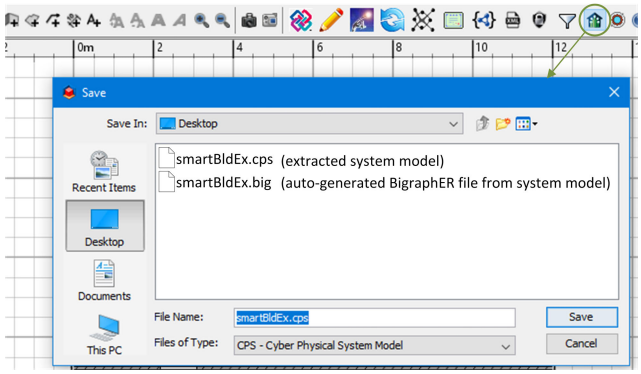

FIGURE 9. Example output of functionality *extract system model*.

behavior of a smart building. Actions can represent a person entering a room or connecting his/her laptop to a computing device via the bus network. The screen for editing actions is shown in Fig. 8. Panel *System Actions* describes the actions currently added to the system; *Action Details* allows visualizing, editing and removing a selected system action; *Actions Catalog* shows the pre-defined actions that could be added to the system. An Action is expressed as a re-writing rule where a portion of the system matching a *pre-condition* is re-written with the sub-system represented in the *post-condition*. Pre- and post-conditions are expressed using a custom notation inspired by Bigraphical Reactive Systems (BRS) [24], which allows representing cyber and physical components and their connectivity and containment relations. Creating a new action requires a security administrator to specify a name and a pre- and post-condition. For example, Fig. 8 shows action "*enter Room*". The pre-and post-condition indicate that two different rooms ($Room_1$ and $Room_2$) are contained in the same physical structure (see operator '|'), for example, the same floor. An *Actor* is inside $Room_1$ (see operator '.'). As a result of the action execution, the *Actor* who was previously contained in $Room_1$ is now inside $Room_2$.

4) *Extract System Model*. Finally, it is possible to use functionality *extract system model* to convert the graphical representation of the smart building to an Eclipse Modeling Framework (EMF)[2] model representing the structure of the building and its possible actions. This model is compliant with the meta-model we propose in previous work [10] to represent smart buildings, and can be used to represent an incident using the *Incident Editor*.

We also generate a representation of the dynamic behavior of the smart building as a Labeled Transition System (LTS). To do this, we transform the structure of the building to a bigraph [25] and the actions defined for the smart building to a set of reaction rules. We then use BigraphER [26], a software tool that implements bigraphs and their dynamics, to generate the LTS automatically. The representation of the dynamic behavior of a smart building is used to create potential instantiations of an incident pattern within that smart building.

B. INCIDENT EDITOR

The *Incident Editor* allows creating a representation of an incident that can be shared across different organizations. Our incident representation is inspired by *Crime Script* [27], a concept that is used in criminology to describe physical crimes, such as subway mugging. We extend the original use of *Crime Script* to represent incidents that can occur in

[2]EMF: https://www.eclipse.org/modeling/emf/

smart spaces. These incidents can target and exploit cyber and physical components and their interplay. An incident is represented as a sequence of activities; each activity indicates the execution of an action in the smart space. Incident activities may be malicious because they exploit a specific vulnerability, and/or may be legitimate (e.g., enter a room, connect to network). Using our tool, incidents can refer to components and actions defined for one or more smart building models created using the *System Editor*.

The main screen of the *Incident Editor* is shown in Fig. 10. The main panel shows the model of the incident. This always includes entity *CrimeScript*, which is the root entity containing all the incident activities. A *CrimeScript* can optionally be characterized by the following attributes: *Intent* defines the aim of the incident (e.g., disruption); *Motive* defines the motive for the incident (e.g., competitive advantage, revenge); *Goal* specifies the set of goals that are required to achieve the *Intent* (e.g., reach a target location, send malware to a device, runaway).

The Palette (right side of Fig. 10) shows the elements that can be dragged and dropped onto the main panel, to model an incident. They are divided into the following categories: *Incident Entity*, *Entity Relations*, *Activity Element*, *Condition Element*, and *Ordering*.

Category *Incident Entity* allows creating entities that can be referred to in an incident model and can be of one of the following types: *Actor*, *Resource*, *Asset*, and *General*. An *Actor* represents the initiator of an incident activity; it can also be further specified as an *Offender* or a *Victim*. A *Resource* is a tool (software or hardware) that can optionally be used to perform an activity. Specific *Resource*s can be defined in the editor, such as *Laptop* and *Malware*. An *Asset* represents a component in the smart building that is exploited or targeted in an incident activity. Each *Asset* refers to components of a smart building where the incident supposedly occurred. These components can be selected using functionality *Select Asset from System*. *General* allows representing a component that is relevant to the incident, but not included in the representation of the smart building where the incident primarily occurred.

Category *Entity Relation* has *Contain* and *Connect* functionalities that can be used to create, respectively, containment and connectivity relations between incident entities. These relations are relevant to the incident, because they can enable occurrence of some of the incident activities.

Category *Activity Element* can be used to define scenes, activities and their attributes. A *Scene* represents a phase of an incident (e.g., preparation, execution). Each *Scene* contains one or a sequence of activities. An *Activity* represents the execution of an action, and can be characterized by the following attributes. *Initiator* refers to an *Actor*, *Victim*, or an *Offender* that performs the activity. *Target Asset* refers to an *Asset* that is targeted or exploited by the activity. *Resource Used* refers to a *Resource* that is used in the activity. *Location* represents the location where the activity is performed. It can be an *Asset*, which refers to a component in an existing smart building model, or a *General* incident entity not defined previously. *Goal* represents the goal of an activity and should refer to one of the goals specified as an attribute of the *CrimeScript*.

Precondition and *Postcondition* are also attributes of an activity that express the state of the smart building, respectively, before and after the activity execution. An activity can represent the execution of an action defined in the model of the smart building where the incident primarily occurred. In this case, *Precondition* and *Postcondition* represent, respectively, the pre- and post-condition of the selected action. To associate an incident activity with the action of interest from the model of the smart building, functionality *Select Action from System* should be used. Then, entity types specified in the pre- and post-conditions of the selected action should be mapped to incident entities specified in the incident model. Considering our incident example shown in Fig. 1, to represent movement of the offender from the *Hallway* to the *Toilet*, action "*enter Room*" shown in Fig. 8 should be imported. $Room_1$ and $Room_2$ have to be instantiated to assets that refer, respectively, to the *Hallway* and the *Toilet* in the smart building model. *Actor* should be instantiated to the *Offender* defined in the incident model.

An activity *Precondition* and *Postcondition* can also be specified from scratch. To represent the subsystem involved in a pre- or post-condition, a security administrator should select a set of incident entities and specify a set of containment and connectivity relations between them. The incident entities involved in the pre- and post-conditions can be selected using functionality *Select Incident Entity* in category *Condition Element*. Containment and connectivity relations can be created using the functionalities included in category *Entity Relation*.

Category *Ordering* provides elements for defining the order of *Scenes* and *Activities*. *Next Scene* allows defining a chronological order between two subsequent incident scenes. Similarly, *Next Activity* is used to define a chronological order between two subsequent activities.

An incident model can be created by performing the following operations.

- *Edit CrimeScript*. When a new incident model is created, it is necessary at least to provide a *name* for the *CrimeScript*. *Intent*, *Motive*s, and *Goal*s can also be specified in natural language to better describe an incident, although it is not mandatory. As shown in Fig. 11, for our incident example, the *name* of the *CrimeScript* is set to "disable hvac".
- *Add IncidentEntiti*es.
  Actor, *Asset*, *Resource*, or *General* incident entities can be subsequently added to the model. If an *Asset* is added to the model, it is necessary to select it explicitly from the model of the smart building, using function *Select Asset from System*. The Properties view (bottom of Fig. 10) displays the properties of a selected entity in the incident model. From there, a security administrator can view and change these attributes —if necessary.

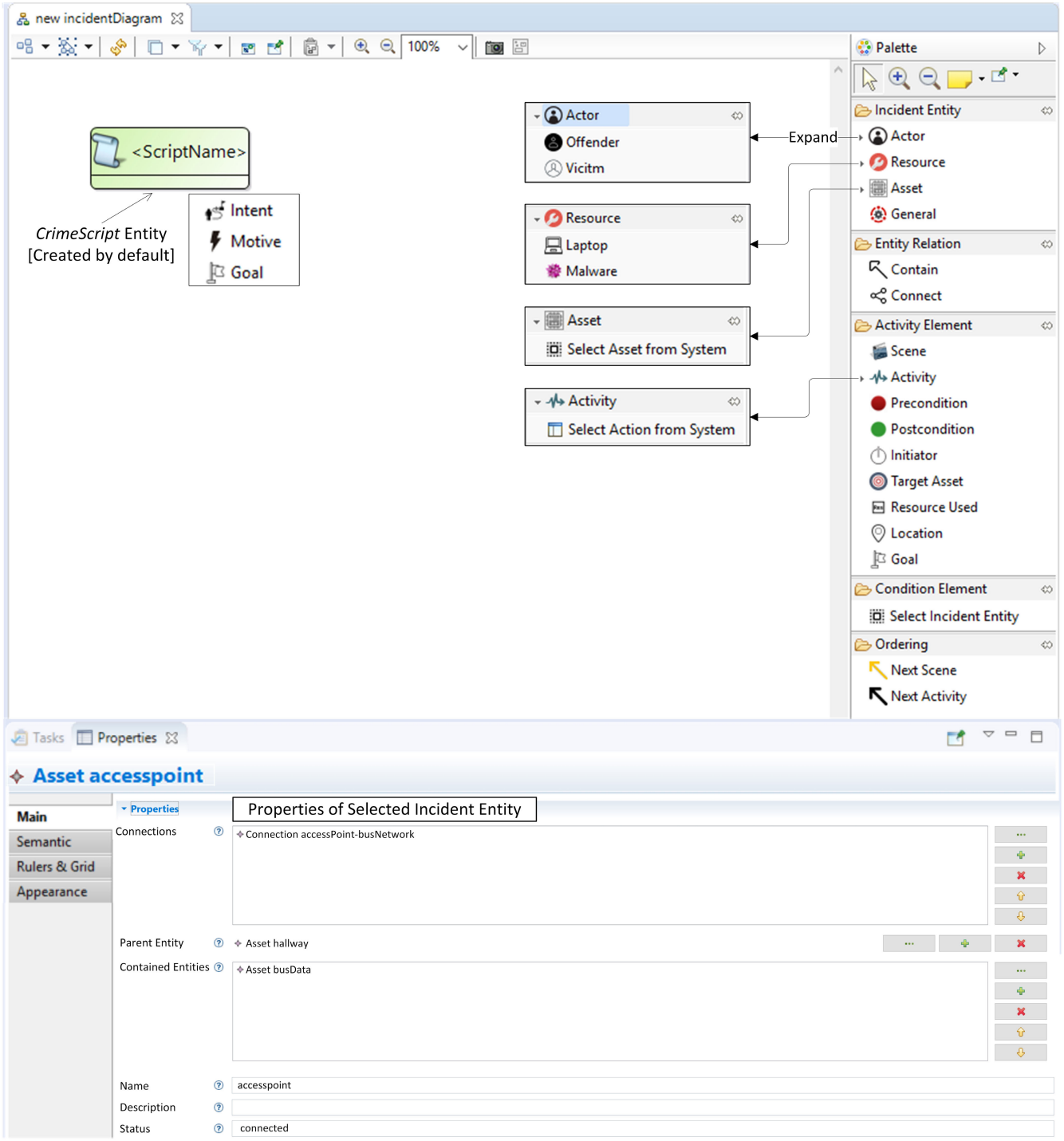

FIGURE 10. ***Incident editor*** **main screen.**

Relations between incident entities can also be specified using functionality *Contain* and *Connect*.

For example, as shown in Fig. 11, a security administrator adds the following *Asset*s (1): *secondFloor*, *toilet*, and *sl1*. Fig. 11 shows how *sl1* can be selected from the model of the smart building using functionality *Select Asset from System*. After that, the security administrator creates an *offender* and a *laptop*. Then s/he creates relations between incident entities (2): connectivity between *sl1* and the *laptop* and containment between *offender* and *laptop*, i.e. the offender carries the laptop. Finally, the security administrator can edit the properties of some of the incident entities. For example, s/he can edit the name, description and status of the *laptop* (3).

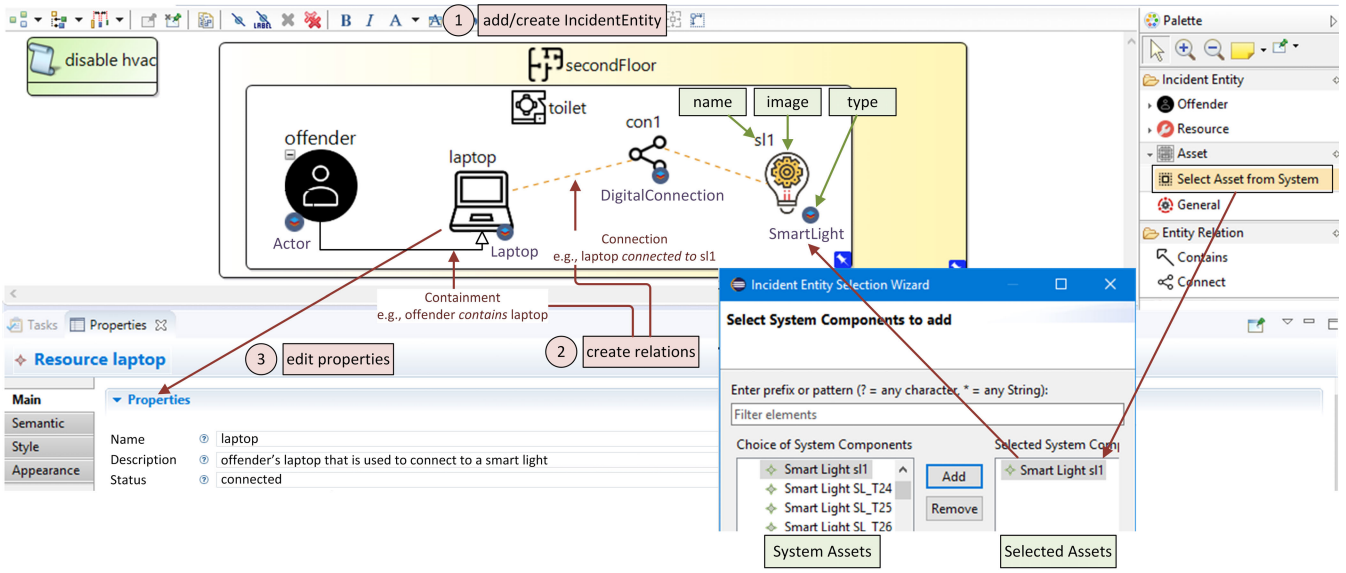

FIGURE 11. Creating *incidententities* and their relations.

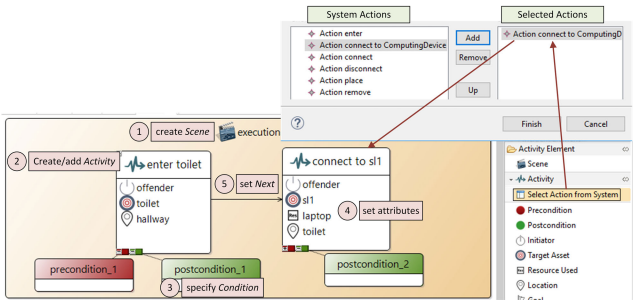

FIGURE 12. Creating *scenes*, *activities*, and their attributes.

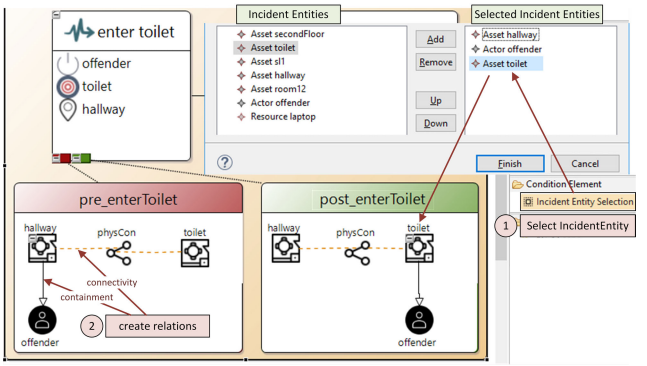

FIGURE 13. Creating a *condition*.

- *Create Activity*. A security administrator should create the incident *Scene*s and, then, s/he can add relevant activities to the scenes. For example, as shown in Fig. 12, a security administrator first defines a *Scene* named "execution" (1). After that, the s/he creates activities "enter Toilet" and "connect to SL1" (2). These activities represent the execution of actions "enter room" and "connect to ComputingDevice" in the model of the smart building. Fig. 12 shows how action *connect to ComputingDevice* can be selected from the model of the smart building using functionality *Select Action from System*. Subsequently, the security administrator defines activities *Precondition* and *Postcondition*, which will be illustrated in more detail in the next point (3). Then s/he defines the activities' attributes (4), such as *Initiator*, which —in this case— is the *offender* for both activities. Finally, s/he indicates that *Activity* "enter toilet" precedes *Activity* "connect to SL1" using functionality *Next Activity* (5).
- *Define Condition.* As shown in Fig. 13, to define an activity *Precondition* and *Postcondition* a security administrator adds the incident entities involved in the pre- and post-conditions, using functionality *Select Incident Entity* (1). Fig. 13 shows how asset *toilet* can be selected from the model of the incident using functionality *Select Incident Entities*. Then, the security administrator defines the relations (i.e. *Containment* and *Connectivity*) between the involved incident entities (2). Fig. 13 shows the *Precondition* and *Postcondition* for *Activity* "enter toilet". In the *Precondition*, the security administrator defines a containment relation between the *hallway* and the *offender* i.e. hallway *contains* offender. S/he also defines a connectivity relation between the *hallway* and the *toilet*. In the *Postcondition*, the security administrator changes the containment to become *toilet contains offender*.

To further help security administrators in modeling an incident, we provide *filters* and *validations* functionalities in the *Incident Editor*. Filters can be used to hide certain aspects

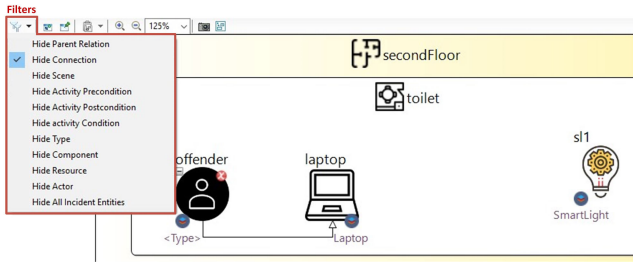

FIGURE 14. Filters in the incident editor, with the "hide connection" filter selected.

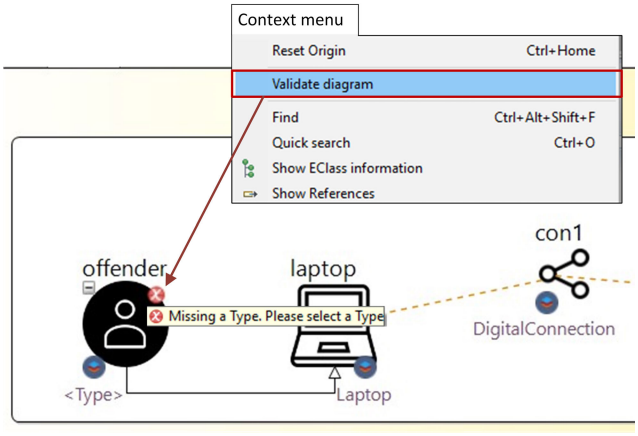

FIGURE 15. Validating an incident model. Validation highlights a missing *type* for the *actor* "offender".

of an incident model that do not require to be visualized. For example, one can hide all *Connections* between entities, or hide a certain type of incident entities, such as *Actor*s. Available filters in the *Incident Editor* are shown in Fig. 14. Validation functionalities check for syntactical correctness of the incident model. For example, a security administrator may forget to specify a *type* of an *IncidentEntity*. In this case, when validating the model, the *Incident Editor* shows an error message ("Missing a Type. Please select a Type.") attached to the *IncidentEntity*, as shown in Fig. 15. Other validations include *Activities* in-sequence (i.e. only the last activity in a scene should not be associated with the next one), *Condition* not-empty, and *Containment* no-cycle (i.e. an *IncidentEntity* cannot contain an *IncidentEntity* that is one of its containers).

VI. IMPLICATIONS ON SHARING AND ASSESSMENT

A security administrator can share information about security incidents and also assess whether and how incidents can re-occur —even in a different smart space. In the rest of this Section we describe how our *Incident Editor* and *Incident Filter* can be used to support sharing of incident knowledge and incident assessment.

A. SHARING INCIDENT KNOWLEDGE

Information about a security incident can be shared by performing the following steps.

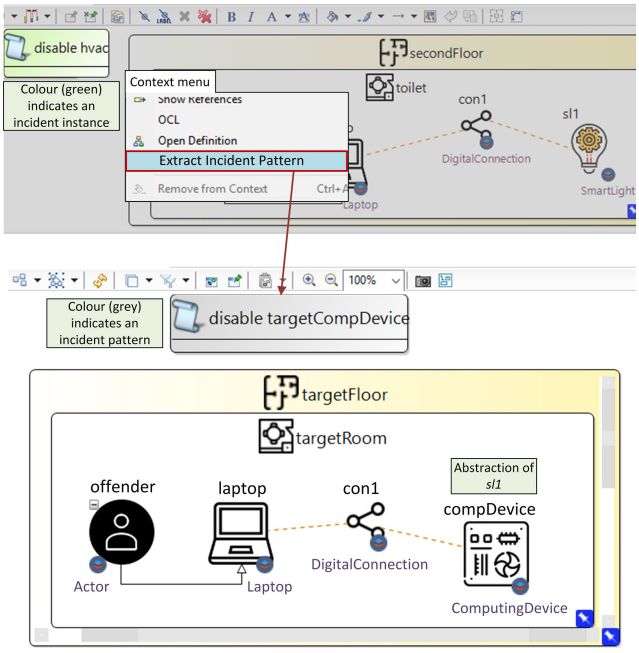

FIGURE 16. Output of function *extract incident pattern* for the incident instance shown in Fig 12.

1) *Generate an incident pattern*. After an incident model is created, a security administrator can extract an incident pattern using functionality *Extract Incident Pattern* in our *Incident Editor*, as indicated in Fig. 16. An incident pattern is a more abstract representation of a security incident that avoids disclosing confidential information. To generate an incident pattern we reuse an automated extraction technique that we proposed in previous work [10]. This technique implements two heuristics. One heuristic abstracts the type of the assets referred by the incident entities defined in the incident model. More precisely, assets are only described using a type that is at one level of abstraction higher than their original type in the smart building meta-model. For example, smart light *sl1* is simply referred to as a *ComputingDevice*, which abstracts type *SmartLight*, as shown in Fig. 16. Another heuristic is to merge sequences of incident activities that match an activity pattern. An activity pattern is a compressed representation of a recurring activity that can be observed in many security incidents. We define activity patterns based on the Common Attack Pattern Enumeration and Classification (CAPEC) catalog [15].
2) *Review the incident pattern*. A security administrator can view the generated incident pattern using the *Incident Editor*. S/he can also edit the incident pattern directly, or modify the original incident instance in order to regenerate a new pattern. This allows a security administrator to change the incident pattern, for example, to make sensitive components more abstract and/or to remove sensitive components and/or activities.

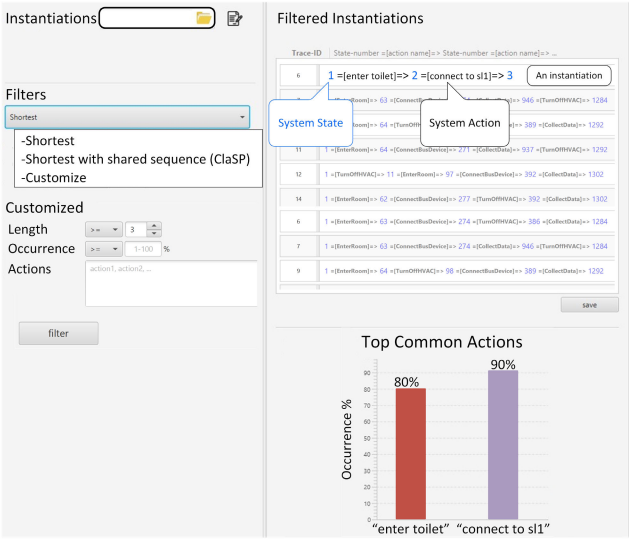

FIGURE 17. Main screen of the *incident filter* component.

3) *Share the incident pattern*. When the security administrator is satisfied with the generated incident pattern, s/he can share it by uploading it to the *Incident Pattern Repository*.

B. INCIDENT ASSESSMENT

A security administrator can use the *Incident Editor* to assess whether and how incident patterns can re-occur in smart buildings. S/he can do so by performing the following steps.

1) *Instantiate the incident pattern*. A security administrator has to first select an incident pattern from the repository. S/he should also select the smart building where the incident pattern should be instantiated, by importing the files representing the building structure and dynamic behavior (see Fig. 9). Then, a security administrator can use functionality *Instantiate Incident Pattern* in the *Incident Editor*, to generate all possible instantiations of an incident pattern. To do so, we reuse an automated instantiation technique that we proposed in previous work [10]. This technique analyzes the LTS modeling the dynamic behavior of the selected smart building, to identify traces that match the activities represented in the incident pattern. A matching instantiation should be composed of sub-instantiations, each of them, matching the activities of the incident pattern in the same temporal order. An activity matches a sub-instantiation of the LTS when the first state of the sub-instantiation satisfies the activity pre-condition and the last state of the sub-instantiation satisfies the activity post-condition.
2) *Filter instantiations*. A security administrator can also use our *Incident Filter* component to prioritize the most relevant incident instances. The main screen of the *Incident Filter* is shown in Fig. 17. After the incident instantiations are provided as input, the *Incident Filter* identifies the minimum and maximum length of incident instantiations, i.e. the minimum and maximum number of actions of which are composed the LTS traces representing the incident instantiations. The *Incident Filter* also identifies actions with their occurrences, i.e. number of times an action appears in the instantiations. Then, a security administrator can select an appropriate filter to prioritize incident instantiations. Current available filters include *Shortest*, *Shortest with shared sequence (ClaSP)*, and *Customized*. *Shortest* filter identifies shortest instantiations, which are relevant because they require a minimum number of actions to be performed. *Shortest with shared sequence (ClaSP)* filter identifies instantiations that share the longest partial sequence of actions and have a minimum number of actions. This filter can identify incident instantiations that are only composed of actions that are strictly necessary for an incident to be successful, while avoiding to include unnecessary actions. This filter uses a sequential pattern mining technique called ClaSP [28]. An implementation of the ClaSP algorithm is provided by SPMF,[3] which is an open source data mining library. *Customized* filter allows a security administrator to customize filters by tweaking various properties, such as the maximum length of the desired sequence, actions that an incident instantiation should include, and minimum percentage of instantiations in which each action occurs at least once.

The right-top side of the *Incident Filter* shows the filtered incident instances. An instantiation is represented as a sequence of states and actions; it has the form of "system-state-number =[action-name]=> system-state-number ...". A *system-state-number* refers to a system state in the LTS representation of the dynamic behavior of the smart building. *Action-name* indicates the system action that moves the smart building to a new state. For example, as indicated in Fig. 17, the first filtered instantiation has the sequence "1 =[enter toilet]=> 2 =[connect to sl1]=> 3". For example, state 1 represents the smart building when the offender is in the hallway. The right-bottom side of Fig. 17 shows the *Top Common Actions*, i.e. the actions that occur in the majority of the instantiations. For example, "enter toilet" and "connect to sl1" are the top common actions, with occurrence percentages of 80% and 90%, respectively, in all filtered instantiations.

VII. CASE STUDY

To evaluate our approach, we use a case study inspired by a real-world incident scenario [29]. The case study revolves around an offender who was able to obtain information about relevant assets in a smart building, by eavesdropping packets transmitted over the bus network. This security incident was

[3] http://www.philippe-fournier-viger.com/spmf/index.php

FIGURE 18. A model of the research center floor that we created using the *System editor*.

facilitated by the limited presence of security features in the bus network protocol (e.g., KNX [30]). We use our approach to model and share information about the security incident of the case study.

We assume the incident to have occurred in one of the floors of a research center to which we had access. We model the research center floor, rooms, components, and behavior using the *System Editor*. We modify the original floor layout by populating it with various smart devices (e.g., smart lights) to mimic a real smart building setup. The floor includes 90 components and 30 actions; the LTS representing the dynamic behavior of the smart building is composed of 30K states and 110,569 transitions.

Then, we model the incident activities using the *Incident Editor*. The security incident consists of 5 activities: *enterFloor*, *enterHallway*, *connectToAccessPoint*, *connectToBusNetwork*, and *collectDataFromBusNework*. Then we extract an *incident pattern* and instantiate it to the same smart building to identify all possible ways in which the incident can re-occur. Finally, we use the *Incident Filter* to prioritize incident instantiations. The tools and data are available online.[4] We measure the overhead (time) required for modeling, and discuss the advantages and limitations of our approach.

A. RESULTS

Modeling the layout and components of the floor (90 components) of our case study using the *System Editor* took around 65 minutes. The modeled floor is shown in Fig. 18. Modeling actions of the system (30 actions) took around 130 minutes. Each action took around 3-5 minutes to model it.

Modeling the incident of our case study using the *Incident Editor* took approximately 85-95 minutes. Modeling an activity took 10-15 minutes. Fig. 19 shows the modeled incident, which includes three scenes: *movingAround*, which has two activities (*enterFloor*, *enterHallway*); *establishingConnection*, which has two activities (*connectToAccessPoint*, *connectToBusNetwork*); and *collectingData*, which has one activity (*collectDatafromBusNetwork*).

243,928 incident instantiations were generated in total. Using the *Incident Filter*, we identified 368 instantiations to be the shortest. This number is further reduced by using the sequential pattern mining technique ClaSP, which allows us to select 120 instantiations, as shown in Fig. 20. The *Incident Filter* also identifies three actions as the top common actions (*EnterRoom*, *ConnectBusDevice*, and *CollectData*) in all the instantiations, as shown in Fig 21.

B. DISCUSSION

The time for modeling a smart building may vary depending on its size, i.e. the number of rooms, components, and connections between them. It also depends on the number of actions that the system should include. Another limitation is that modeling actions requires having knowledge about Bigraphical Reactive Systems [25] and the syntax used by BigraphER [26] to represent pre- and post-conditions. We mitigate this limitation by creating an *Actions Catalog*, which contains various actions that are common in a smart building such as ''connect_to_computingDevice'', and

[4]https://sourceforge.net/projects/tools-incidents-smart-spaces/files/

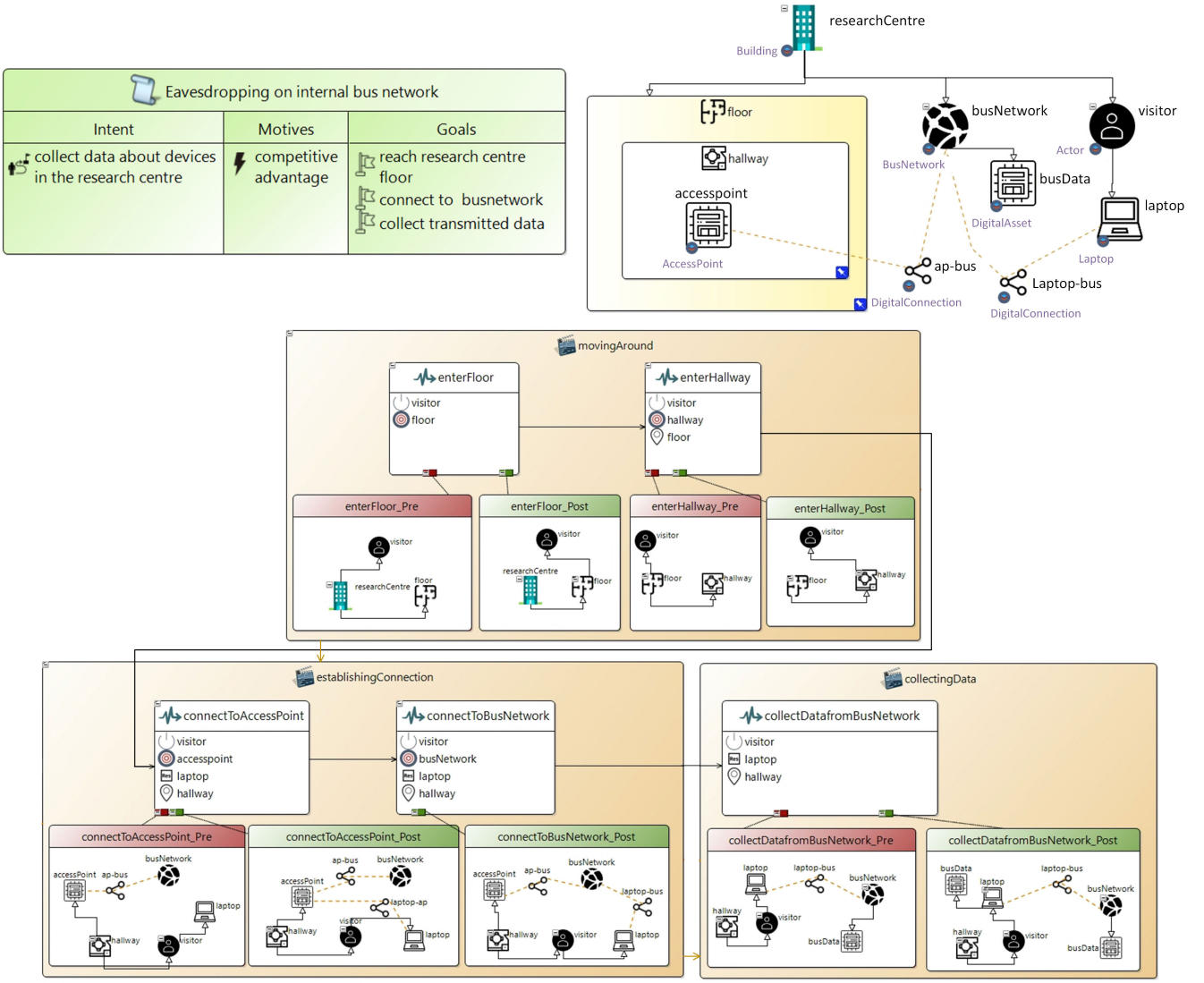

FIGURE 19. **A model of the research center incident that we created using the *incident editor*.**

"enter_room". These actions can be selected by a security administrator directly without the need to represent them from scratch. Also, a security administrator can only select the components that are available in the *System Editor* to represent a smart building. However, we can easily add components to the *System Editor* by extending the smart building meta-model with the component type, and then importing a 3D model of the component to the *System Editor*.

The time for modeling an incident may vary depending on the number of components and activities that are relevant to the incident. We mitigate this limitation by allowing security administrators to refer to components and actions from a selected smart building model, when defining an incident. For our case study, one of the authors of this paper performed the modeling activities. Thus, the times recorded to model the smart building and the incident instance could increase in case a non-expert adopts our tool. In future work, we are planning to perform an extensive evaluation of the tool, assessing its usability with external users and domain experts.

We discussed expressiveness of our approach for incident reporting with a practitioner working on access control for smart buildings. He agreed on the fact that the proposed representation allows modeling explicitly vulnerabilities arising from the interplay of cyber and physical components of a smart building. The practitioner also agreed on the importance and the relevance of the filters implemented to reduce the number of incident instances that a security administrator has to examine. However, one limitation of our approach is that it only allows modeling time-discrete activities, but cannot express time-continuous activities, for example to characterize the behavior of a physical component (e.g., power plant). Furthermore, activities pre- and post-conditions can only refer to qualitative relations (connectivity and containment) between smart building components, and cannot model

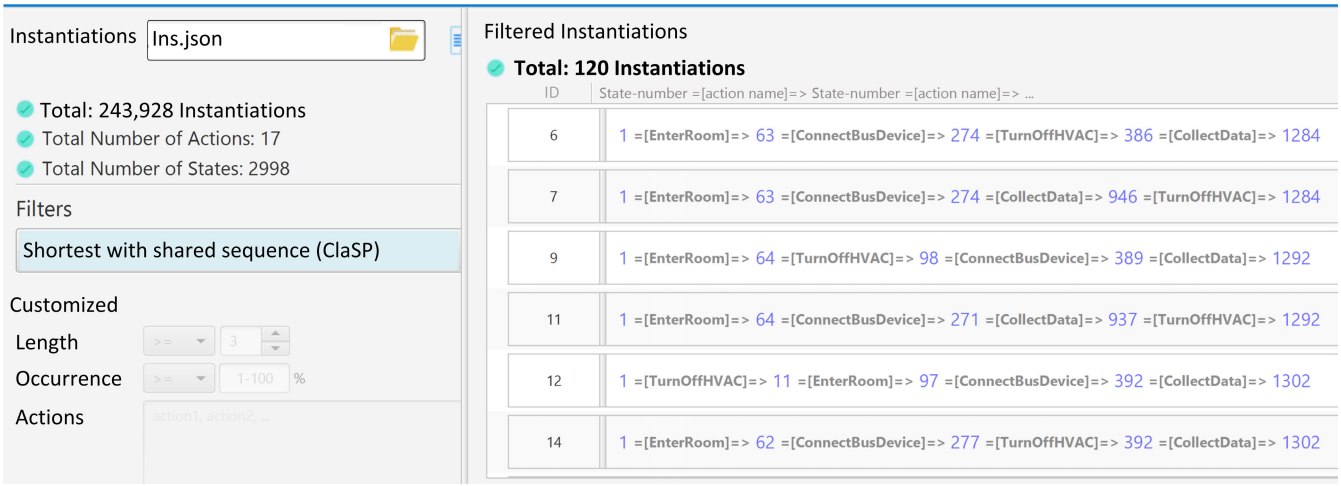

FIGURE 20. Filtered instantiations using ClaSP.

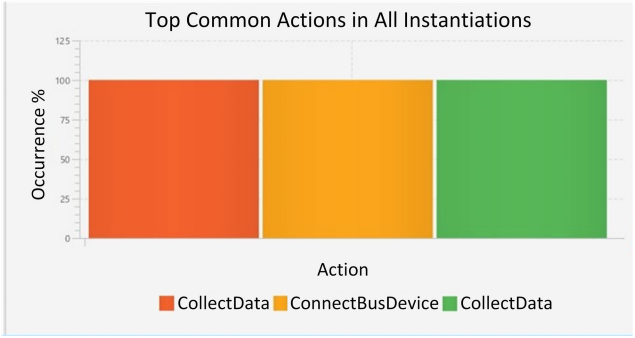

FIGURE 21. Top common actions in all generated instantiations.

quantitative relationships (e.g., an actor is 5 meters away from a device). However, using the types of activities proposed in our approach we were able to represent some of the most recent security incidents occurring in smart buildings, such as the German Still mill [2] and the Ukrainian Power Grid [3] incidents.

VIII. CONCLUSION & FUTURE WORK

In this paper, we discussed limitations in reporting security incidents in smart spaces. To address these limitations, we proposed a set of software tools that can be used to automate the modeling of smart spaces and the reporting of incidents, respectively. We discussed the design, functionalities, and use of a *System Editor* to represent smart spaces. We presented an *Incident Editor* and illustrated how it can be used to report an incident in a smart space via a case study. Using our previous work, incidents represented using our editor can be shared and instantiated in smart spaces. Since an incident can be realized in many ways in a smart space, we proposed an *Incident Filter* that can be used to view and prioritize the most relevant incident instantiations. We suggest that our work represents an important first step in automating the reporting of security incidents in smart spaces.

In future work, we will extend the functionalities of the *Incident Editor* to facilitate the modeling activities. For example, we will suggest potential actions/components that can be relevant to the incident that is being modeled. We will enrich the system actions catalog and facilitate the representation of actions' pre- and post-conditions by providing a more intuitive GUI. We are also planning to assess expressivity of our modeling approach, by using it to model a large set of security incidents in smart buildings. Furthermore, we will assess usability of the tool with domain experts. Finally, our ultimate objective is to use our approach for automated incident reporting to make smart spaces more secure and forensic-ready [31]. To achieve this aim, we will use potential incident instantiations to suggest appropriate security controls and to identify monitoring activities that can be useful to detect and/or investigate these potential incidents.

REFERENCES

[1] A. A. Cárdenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. Sastry, "Challenges for securing cyber physical systems," in *Proc. Workshop Future Directions Cyber-Phys. Syst. Secur.*, vol. 5, 2009, pp. 1–7.
[2] R. M. Lee, M. J. Assante, and T. Conway, "German steel mill cyber attack," *Ind. Control Syst.*, vol. 30, p. 62, Dec. 2014.
[3] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber attack on the Ukrainian power grid," Electr. Inf. Sharing Anal. Center, Washington, DC, USA, Tech. Rep., 2016.
[4] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide," Nat. Inst. Standards Technol., Tech. Rep., 2012. Accessed: May 11, 2018. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-61rev2/SP800-61rev2.pdf
[5] N. Tulechki, "Natural language processing of incident and accident reports: Application to risk management in civil aviation," Ph.D. dissertation, CLESCO, Univ. Toulouse le Mirail-Toulouse II, Toulouse, France, 2015.
[6] M. J. West-Brown, D. Stikvoort, K.-P. Kossakowski, G. Killcrece, and R. Ruefle, "Handbook for computer security incident response teams (CSIRTs)," Softw. Eng. Inst., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 2003. Accessed: Jun. 15, 2018. [Online]. Available: http://www.dtic.mil/docs/citations/ADA413778
[7] H. S. Lallie, K. Debattista, and J. Bal, "An empirical evaluation of the effectiveness of attack graphs and fault trees in Cyber-attack perception," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1110–1122, May 2018.

[8] MITRE Corporation. *Common Vulnerabilities & Exposures (CVE)*. Accessed: Nov. 15, 2018. [Online]. Available: https://cve.mitre.org/
[9] RISI. *The Repository of Industrial Security Incidents (RISI)*. Accessed: Nov. 15, 2018. [Online]. Available: http://www.risidata.com/
[10] F. Alrimawi, L. Pasquale, D. Mehta, N. Yoshioka, and B. Nuseibeh, "Incidents are meant for learning, not repeating: Sharing knowledge about security incidents in cyber-physical systems," Jun. 2019, *arXiv:1907.00199*. [Online]. Available: https://arxiv.org/abs/1907.00199
[11] M. Chau, J. J. Xu, and H. Chen, "Extracting meaningful entities from police narrative reports," in *Proc. Annu. Nat. Conf. Digit. Government Res.*, 2002, pp. 1–5.
[12] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *Proc. Int. Symp. Formal Methods Compon. Objects*. Berlin, Germany: Springer, 2004, pp. 344–371.
[13] *Standardizing Cyber Threat Intelligence Information With the Structured Threat Information Expression (STIX)*, MITRE Corporation, McLean, VA, USA, 2012, vol. 11, pp. 1–22.
[14] OASIS Open. *Introduction to TAXII*. Accessed: Jul. 10, 2018. [Online]. Available: https://oasisopen.github.io/cti-documentation/taxii/intro
[15] MITRE Corporation. *Common Attack Pattern Enumeration & Classification*. Accessed: Jan. 20, 2018. [Online]. Available: http://capec.mitre.org/
[16] NVD. *The US National Vulnerability Database (NVD)*. Accessed: Jan. 20, 2018. [Online]. Available: https://nvd.nist.gov/
[17] B. Potteiger, G. Martins, and X. Koutsoukos, "Software and attack centric integrated threat modeling for quantitative risk assessment," in *Proc. Symp. Bootcamp Sci. Secur. (HotSos)*. New York, NY, USA: ACM Press, 2016, pp. 99–108.
[18] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication systems for building automation and control," *Proc. IEEE*, vol. 93, no. 6, pp. 1178–1203, Jun. 2005.
[19] T. Mundt and P. Wickboldt, "Security in building automation systems—A first analysis," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)*, 2016, pp. 1–8.
[20] B. Krebs. (2016). *IoT Reality: Smart Devices, Dumb Defaults*. [Online]. Available: https://krebsonsecurity.com/2016/02/iot-reality-smart-devices-dumb-defaults
[21] A. Ahmad, J. Hadgkiss, and A. B. Ruighaver, "Incident response teams—Challenges in supporting the organisational security function," *Comput. Secur.*, vol. 31, no. 5, pp. 643–652, 2012.
[22] L. Pasquale, C. Ghezzi, E. Pasi, C. Tsigkanos, M. Boubekeur, B. Florentino-Liano, T. Hadzic, and B. Nuseibeh, "Topology-aware access control of smart spaces," *Computer*, vol. 50, no. 7, pp. 54–63, 2017.
[23] C. Eastman, P. Teicholz, R. Sacks, and K. Liston, *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. Hoboken, NJ, USA: Wiley, 2011.
[24] R. Milner, "Pure bigraphs: Structure and dynamics," *Inf. Comput.*, vol. 204, no. 1, pp. 60–122, 2006.
[25] R. Milner, *The Space and Motion of Communicating Agents*, 1st ed. New York, NY, USA: Cambridge Univ. Press, 2009.
[26] M. Sevegnani and M. Calder, "BigraphER: Rewriting and analysis engine for bigraphs," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2016, pp. 494–501.
[27] D. Cornish, "Crimes as scripts," in *Proceedings of the International Seminar on Environmental Criminology and Crime Analysis*. Tallahassee, FL, USA: Florida Criminal Justice Executive Institute, 1994, pp. 30–45.
[28] A. Gomariz, M. Campos, R. Marin, and B. Goethals, "ClaSP: An efficient algorithm for mining frequent closed sequences," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*. Berlin, Germany: Springer, 2013, pp. 50–61.
[29] W. Granzer, F. Praus, and W. Kastner, "Security in building automation systems," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, pp. 3622–3630, Nov. 2010.
[30] H. Merz, T. Hansemann, and C. Hübner, *Building Automation*. Cham, Switzerland: Springer, 2009.
[31] L. Pasquale, D. Alrajeh, C. Peersman, T. Tun, B. Nuseibeh, and A. Rashid, "Towards forensic-ready software systems," in *Proc. 40th Int. Conf. Softw. Eng., Ideas Emerg. Results (NIER)*, Gothenburg, Sweden, May/Jun. 2018, pp. 9–12.

FAEQ ALRIMAWI received the B.Sc. degree in computer system engineering from Birzeit University, Palestine, in 2010, and the M.Sc. degree in mobile and high-speed telecommunication networks from Oxford Brookes University, U.K., in 2012. He is currently pursuing the Ph.D. degree in computer science with Lero—The Irish Software Research Centre, University of Limerick, Ireland.

His research interests include software engineering, digital forensics, and security for cyber-physical systems.

LILIANA PASQUALE received the Ph.D. degree from the Politecnico di Milano, Italy, in 2011. She is currently a Lecturer with University College Dublin, Ireland, and a Researcher with Lero—the Irish Software Research Centre.

Her research interests include requirements engineering and adaptive systems, with a particular focus on security, privacy, and digital forensics. She has served in the Program and Organizing Committee of prestigious software engineering conferences, such as ICSE, FSE, ASE, and RE. She is also part of the review committee of the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING journal and the *ACM Transactions on Software Engineering and Methodology* journal.

BASHAR NUSEIBEH is currently a Professor of computing with The Open University and a Professor of software engineering and a Chief Scientist with Lero—The Irish Software Research Centre. He is also a Visiting Professor with University College London (UCL) and the National Institute of Informatics (NII), Tokyo, Japan.

Dr. Nuseibeh is a Fellow of the British and Irish Computer Societies, a Fellow of the Institution of Engineering and Technology, and a member of Academia Europaea. He received the ICSE Most Influential Paper Award, the Philip Leverhulme Prize, the Automated Software Engineering Fellowship, the Royal Academy of Engineering Senior Research Fellowship, the IFIP Outstanding Service Award, in 2009, the ACM SIGSOFT Distinguished Service Award, in 2015, the Royal Society-Wolfson Merit Award, and two European Research Council (ERC) awards, including the ERC Advanced Grant on "Adaptive Security and Privacy." He was the Chair of the Steering Committee of the International Conference on Software Engineering (ICSE). He serves as the Editor-in-Chief for the *ACM Transactions on Autonomous and Adaptive Systems* and an Associate Editor of the *IEEE Security and Privacy Magazine*.